Travaux Pratiques d'Interfaçage Logiciel
première séance

ENSIBS 2^{eme} année, Spécialités Mécatronique et Génie Industriel

1 Objectifs

Ce TP s'inscrit dans le cours d'interfaçage Logiciel et fait partie d'une série de deux TP dont l'objectif est de vous familiariser avec du matériel.

Cette première séance doit vous permettre de prendre en main la plate-forme dsPIC et d'apprendre à utiliser des périphériques en utilisant les plus basiques d'entre-eux que sont les *boutons poussoirs* et les *diodes electro luminescentes* (LED).

A l'issue du TP vous devez savoir :
- Configurer et piloter les ports d'Entrée/Sortie
- Utiliser des opérations de *masquage* pour piloter chaque E/S
- Utiliser des symboles
- Gérer une interruption

2 Description du matériel

Les deux scéances de TP se dérouleront sur une carte de développement *dsPIC* sur laquelle sont précablés un ensemble de périphériques. Parmi ceux-ci ont trouve deux liens série RS232, un port CAN, un écran LCD, des entrées/sorties analogiques ainsi que des LED et des boutons poussoir.

Le développement sur la carte qui vous est proposé sera réalisé à l'aide de l'environnement *MPLAB* fourni par Microchip auquel est associé le compilateur C *C30*.

2.1 Implantation des périphériques

Les paragraphes suivants décrivent sur quelles broches du dsPIC les différents périphériques utilisés sont mappés.

2.1.1 LEDs

Les LED sont connectées aux 4 bits de poids faible du port D. Une LED s'allume lorsqu'un 0 logique est émis sur la sortie correspondante.

2.1.2 Boutons Poussoir

Les Boutons poussoirs sont connectés aux 4 bits de poids fort du port A. Lorsque le Bouton Poussoir est appuyé, on peut lire un 0 sur la sortie corrspondante.

3 Réalisations

3.1 Prise en main de la maquette

Il est tout d'abord nécessaire de prendre en main l'utilisation de la maquette. Pour cela, nous allons exécuter le programme C décrit sur le listing 1 et permettant d'allumer une diode sur la maquette.

Listing 1 – Programme de Test

```
1  #include <p30f6014.h>
2
3  void main (void)
4  {
5    // initialisation du port D
6    TRISD=0xFFF0; // Port D en sortie
7    LATD=0x000F;  // Toutes LEDS eteintes
8
9    PORTD &= ~0x0001; // on allume la LED 1
10
11   while (1); // boucle infinie
12 }
```

MPLAB fonctionnant par projet, il est d'abord nécessaire de créer un projet. Une fois le projet créé, on peut commencer à écrire ses programmes. Pour exécuter le programme, il faut passer par une phase de compilation qui est décrite ensuite.

3.1.1 Création d'un projet

La création d'un projet se réalise simplement, en utilisant l'*assistant*.

- Project -> Project Wizard
  - Configurer la cible : *dsPIC30F6014(A)*
  - Choisir la chaîne de développement : *MPLAB C30*
  - Choisissez l'emplacement du projet (je conseille de créer un répertoire pour y mettre tous les fichiers du projet)
  - Ajouter des fichiers au projet
    - Un fichier est nécessaire, il est appelé script d'édition de liens, vous le trouverez à l'adresse : `C:\PROGRAM FILES\Microchip\MPLAB C30\support\gld\p30f6014.gld`.

3.1.2 Écriture du programme

Une fois le projet créé, il faut créer le fichier contenant le programme, écrit en langage C. Ce fichier sera inséré dans le projet.

La procédure de création est la suivante.

- File -> New File
- File -> Save As
- Project -> Add Files To Project

3.1.3 Exécution du programme

Pour pouvoir exécuter le programme, il est nécessaire de passer par plusieurs phases.

La phase de compilation permet de générer le code *objet* qui sera envoyé sur la cible. Chaque fichier C est transformé en un code objet correspondant (fichier portant l'extension *.o*).

L'edition de liens cette phase permet de lier les différents codes objet provenant des sources ou de librairies entre eux. En effet, les différents objets peuvent utiliser des variables ou des fonctions en commun.

La programmation Il s'agit d'envoyer le code sur la cible pour qu'il y soit exécuté. Un *reset* de la cible permettra ensuite de démarrer le programme.

L'outil MPLAB offre une interface intégrant les différentes phases pour exécuter un programme, on effectue donc les étapes suivantes :

- *Project -> Build All* effectue les phases de compilation et d'édition de liens
- *Programmer -> Select Programmer* permet de sélectionner l'interface de programmation (ici *ICD2*).
- *Programmer -> Program* charge le programme dans la cible.
- *Programmer -> Release From Reset* permet d'autoriser l'exécution du code.

3.2 Allumage d'une LED sur un Bouton

Le premier exercice consiste à allumer la *LED1* par appui sur *SW1*.

Deux méthodes sont possibles : on peut utiliser une structure de contrôle conditionnelle *if ... then ... else* ou bien une équation logique reliant la *LED1* au *SW1* à l'aide de masquages.

3.3 Changement d'état d'une LED sur un Bouton

Il s'agit, cette fois-ci de changer l'état de la LED à chaque appui sur le bouton *SW1*.

3.4 Gestion compacte d'un ensemble de LED

En partant de la solution utilisant une équation logique, proposez une solution compacte pour piloter les 4 LED.

3.5 Symbolisation des E/S

Vous avez normalement utilisé jusqu'à présent les *noms des ports* et les masques écrits en *hexadécimal*. Ce n'est pas très explicite, limite la compréhension du code et la portabilité des programmes. C'est pourquoi on symbolise généralement les entrées/sorties.

Pour cela, on définit des macro qui prennent la forme présentée sur le listing 2.

Listing 2 – Définition des symboles

```
1 #define LEDS PORTD
2 #define LED1 0x0001
```

Avant la phase de compilation, tous les appels à `LEDS` seront transformés en `PORTD`. Pour allumer la LED1, on écrira donc le code proposé sur le listing 3

Listing 3 – Allumage d'une LED

```
1 LEDS = LED & ~LED1;
```

Il vous est donc demandé de transformer les programmes précédents pour utiliser les symboles.

Note : Certaines constructions ne sont plus possibles ...

3.6 Gestion par interruption

Il vous est maintenant demandé de gérer l'allumage de la LED1 en utilisant une interruption. Par chance, vous aurez remarqué que le *SW1* est cablé sur la broche *INT1* on va donc pouvoir utiliser l'interruption associée.

Si vous lisez les documentations de Microchip, vous remarquerez que le dsPIC dispose d'une table de vecteurs. Celle-ci est initialisée par le compilateur à partir du nom des fonctions déclarées. Pour *INT1*, ce nom est `_INT1Interrupt`. Il est aussi nécessaire d'indiquer au compilateur que l'on est en présence d'une routine d'interruption, cela est fait à l'aide de la macro `_ISR` qui définit un ensemble de paramètres pour la fonction indiquant notamment au compilateur de générer une instruction `RETFIE`. L'écriture de l'interruption *INT1* prend donc la forme présentée sur le listing 4.

Listing 4 – Utilisation de INT1

```
1 void _ISR _INT1Interrupt (void) {
2
3 // ... Code de l'interruption ...
4
5 }
```

Il sera ensuite nécessaire de valider et configurer l'interruption à l'aide des registres `IECx` et `INTCONx`. L'interruption doit aussi être acquittée au début de la routine en manipulant l'un des registres `IFSx`.

3.7 Utilisation d'un timer

Pour finir, vous allez faire clignoter la LED1 avec une fréquence de $1Hz$ à l'aide d'un *timer*. D'abord sans puis avec l'interruption associée.